Master’s Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
HYBRID IMPLEMENTATION
KIN TAT, TAN

Universitat de Lleida

HYBRID IMPLEMENTATION

KIN TAT, TAN

Universitat de Lleida

Sumário

Figures

Tables

Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
HYBRID IMPLEMENTATION
KIN TAT, TAN

Introduction

In this document will demonstrate the results about the MPI Implementation and the Hybrid Implementation. In the last practice (MPI Implementation) I could not generate good results because I faced several issues and the idea did not match with the algorithm. Throughout in this document I will gonna show the process adopted and explained the reason to decision taken.

GitHub

OpenMP Implementation	https://github.com/tankintat/highperformanceudl/tree/master/OpenMP_Implementations
MPI Implementation	https://github.com/tankintat/highperformanceudl/tree/master/MPI_Implementations
Hybrid Implementation	https://github.com/tankintat/highperformanceudl/tree/master/Hybrid_Implementations

Master’s Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
HYBRID IMPLEMENTATION
KIN TAT, TAN

Universitat de Lleida

OpenMP, MPI and Hybrid

Type	Pro	Cons
OpenMP	- Easier to program and debug than MPI - Directives can be added incrementally - gradual parallelization - Can still run the program as a serial code - Serial code statements usually don't need modification - Code is easier to understand and maybe more easily maintained	- Can only be run in shared memory computers - Requires a compiler that supports OpenMP - Mostly used for loop parallelization
MPI	- Runs on either shared or distributed memory architectures - Can be used on a wider range of problems than OpenMP - Each process has its own local variables - Distributed memory computers are less expensive than large shared memory computers	- Requires more programming changes to go from serial to parallel version - Can be harder to debug - Performance is limited by the communcation network between the nodes

Table 1 - Compared OpenMP and MPI

Hybrid Model

The idea with this model its interconnect the between the multi-nodes SMP (Symmetric Multiprocessor) and each node connect by the shared memory parallel programming spreaded in threads executing the same time.

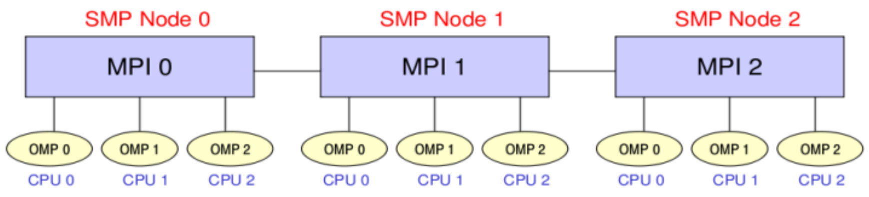

Figure 1- Hybrid illustration

It’s a elegant concept because use OpenMP within in node and MPI between nodes, in order to have a good use of shared resources.

Message Passing Interface (MPI) is a message-passing library specification which is designed for high performance computing in distributed memory systems. In MPI, data is moved from one process (or MPI rank) to another process through cooperative operations. Each MPI rank owns a separate address space. MPI involves both communication and synchronization, and it has become the de facto standard for communication among processes running on distributed memory systems.

Open Multi-Processing (OpenMP) is a specification for a set of pragmas, run-time libraries and environment variables that are used for thread parallelism. Shared memory is accessed by all OpenMP threads. OpenMP programming is used for shared memory processes which run on a multicore processor.

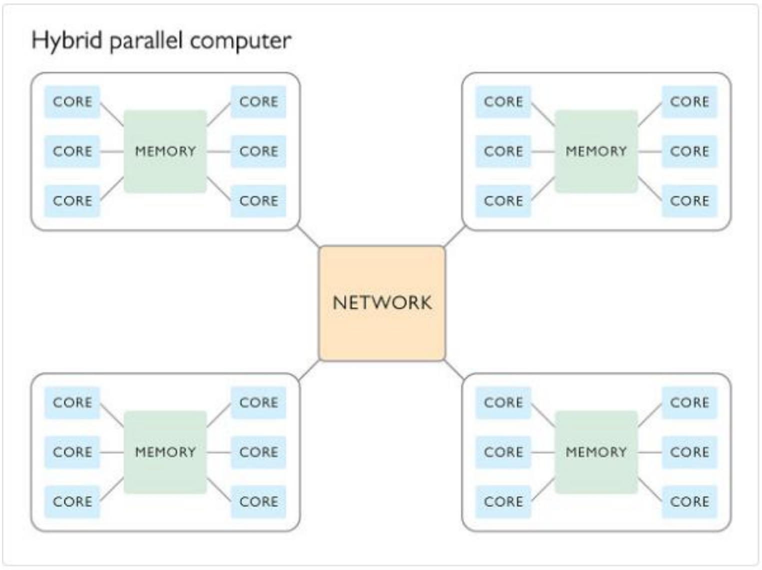

Figure 2 - Hybrid second illustration

Problem Knapsack-MPI Dynamic

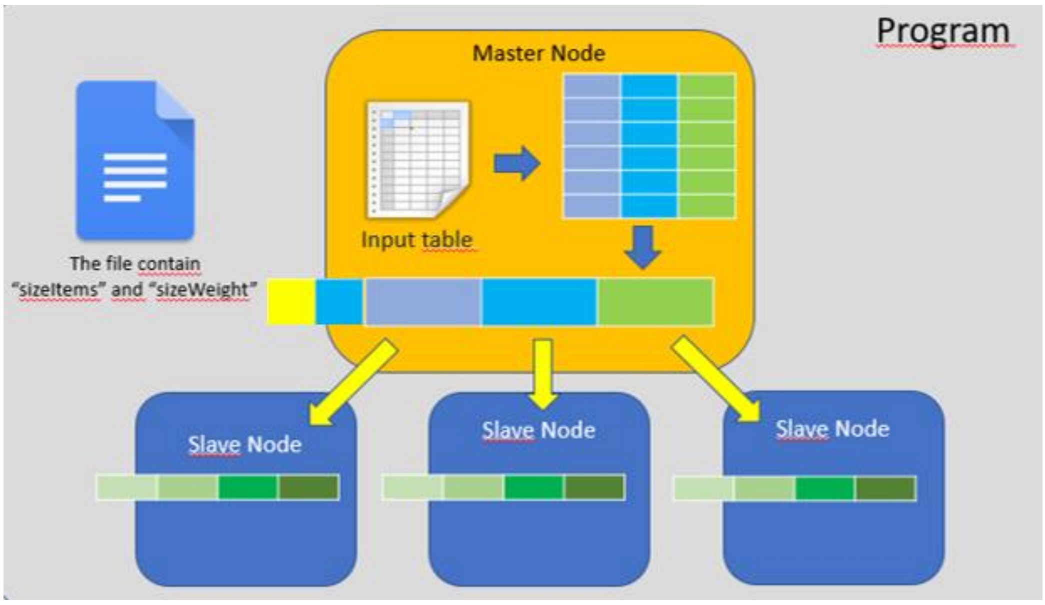

Figure 3 - Idea architecture MPI Implementation

Considering 4 SMP nodes the first we determine the master node and the another nodes just like Slave or Children nodes. The process as follows all the process going to read the file to know the size of the items and the weight. And the master go to read the rest of the file to start the algorithm process.

- Creation the matrix according the knapsack algorithm:
- The first step its put 0 the first line and column in the vector.
- In the master node have to send the previous row in the matrix to process the next one, so in this case we have to determine to each children process which the range to calculate and returns back to the main. So the idea its put the begin information relevant to send the children.
    Start range, End range, Previous value inputted items and the previous value weight items. And the next put the previous line in the matrix.

  And wait the children nodes to send back the results information.
  When the vector arrives put on the matrix the new values.
- In the children nodes will receive that information, each other his own, and process and returns back to the main process.

The reason to choose this way was because the problem Dynamic need the previous row before to process the new one, so I choose the shared in rows by rows.

Problem Knapsack-Hybrid Dynamic

The Hybrid basically use the idea in OpenMP implementation and put the MPI Implementation. The idea its between the machines will send the information in each row and each value in this row we divide in shared data threads to generate the information.

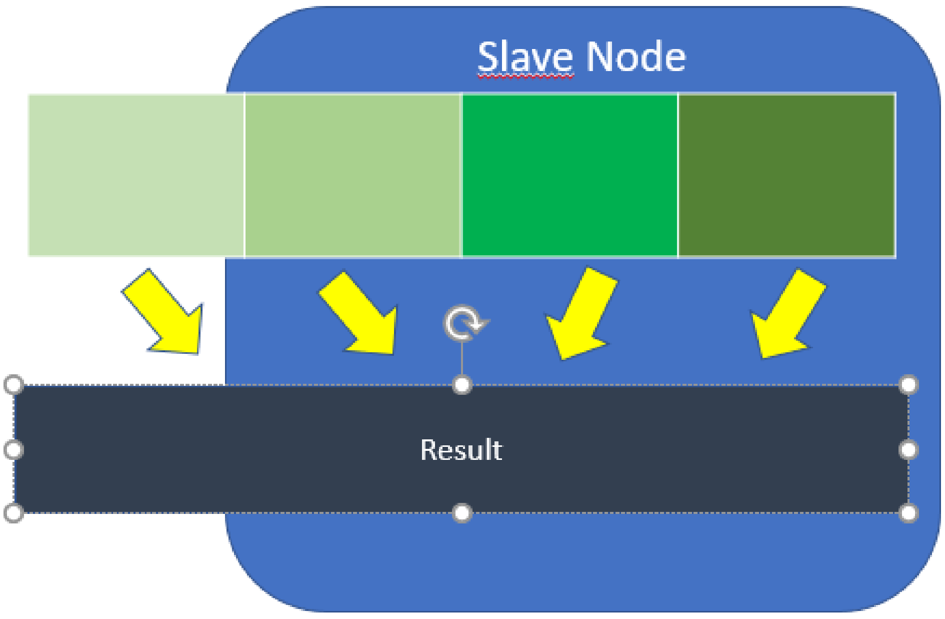

Figure 4 - Architecture Hybrid Implementation

Results MPI Implementation

The results are found in this path on GitHub: mpisource/MPIResults.txt, but in this document, will be separate in order to analyze the data.

Compilation command:

```
mpicc -lm -o knapsackDYN-mpi knapsackDYN-mpi.c
```

- -lm, Math library, some libraries might use.
- -o, output

The execute file found in the: mpisource/run-extended.sh.

Results:

File	Result	Time read file until end	Total time
5000000_100	9900	10.568423	11.793524
5000000_1000	-	-	-
5000000_10000	-	-	-
500000_100	9829	1.055164	1.394754
500000_1000	45660	7.348894	7.639121
500000_10000	8993	0.105615	0.274800
50000_100	-	-	-
50000_1000	16810	0.733503	0.834015
50000_10000	18820	6.548665	6.732827
5000_100	4801	0.014761	0.016989
5000_1000	5296	0.073382	0.210283
5000_10000	5858	0.656423	0.794147
500_100	1782	0.001158	0.196064
500_1000	1722	0.011917	0.012224
500_10000	1728	0.066099	0.221216
50_100	700	0.000304	0.000611
50_1000	533	0.001661	0.001759
50_10000	546	0.012214	0.012471
5_5	129	0.000068	0.000260

Table 2 - Results MPI.

Universitat de Lleida

Results Hybrid Implementation

The results are found in : openmpihybrid/HYBRIDResults8Th.txt

Compilation command:

mpicc -fopenmp knapsackDYN-hybrid.c -lm -o knapsackDYN-hybrid

And inside of the run qsub command file i put the -ppn 1 to determine 1 process per node.

The execute file found in the: openmpihybrid/run-mpi.sh

Results 8 Threads

File	Result	Time read file until end	Total time
5000000_100	-	-	-
5000000_1000	-	-	-
5000000_10000	-	-	-
500000_100	9829	14.225912	14.398057
500000_1000	45660	25.297862	25.427544
500000_10000	-	-	-
50000_100	8993	1.400859	1.416901
50000_1000	16810	2.572228	2.612670
50000_10000	18820	13.273941	13.292517
5000_100	4801	0.151780	0.154081
5000_1000	5296	0.261496	0.263693
5000_10000	5858	1.321893	1.324396
500_100	1782	0.044423	0.044766
500_1000	1722	0.044990	0.045439
500_10000	1728	0.150477	0.150910
50_100	700	0.037209	0.037376
50_1000	533	0.046842	0.046956
50_10000	546	0.023949	0.023949
5_5	129	0.022427	0.022500

Table 3 - Results Hybrid 8 Threads.

Results 16 Threads

The results are found in : openmpihybrid/HYBRIDResults16Th.txt

The same code the previous but change the number of threads.

File	Result	Time read file until end	Total time
5000000_100	-	-	-
5000000_1000	-	-	-
5000000_10000	-	-	-
500000_100	9829	27.171665	27.259739
500000_1000	-	-	-
500000_10000	-	-	-
50000_100	8993	2.645519	2.666395
50000_1000	16810	4.376992	4.391314
50000_10000	-	-	-
5000_100	4801	0.281993	0.284142
5000_1000	5296	0.444889	0.447281
5000_10000	5858	2.066941	2.068759
500_100	1782	0.039452	0.039644
500_1000	1722	0.055688	0.056123
500_10000	1728	0.224018	0.224487
50_100	700	0.014402	0.014503
50_1000	533	0.016513	0.016556
50_10000	546	0.035584	0.035802
5_5	129	0.012949	0.013175

Table 4 - Results Hybrid 16 Thread.

Conclusion

We can conclude the MPI results are most satisfactory for input files the better way is the pure MPI. Process which 50_100 are processed in 0.000611 and a 500x100 process by 0.196064, 5000x100 by 0.016989.

That means para measure each line in the matrix the time will processed:

0.000611/50 = 0.00001222, 0.196064/500 = 0.000392128, 0,016989/5000 = 0.0000033978.

In base this argument we can observe if we increase the iterations we increase the performance each row.

0.284142/5000 = 0.0000568284, 0.447281/5000 = 0.0000894562, 2.068759/5000 = 0.0004137518

Instead if we increase the quantity of the process in the children node, the time will lose performance.

If we change the data the hybrid loses the performance compared with the MPI, and the neck maybe are in OpenMPI.

0.037376/50 = 0.00074752, 0.044766/500 = 0.000089532, 0.154081/5000 = 0.0000308162

If I maintain the data we can observe that Hybrid has increase a little in the process in each node.

0.154081/5000 = 0.0000308162, 0.263693/5000 = 0.0000527386, 1.324396/5000 = 0.0002648792

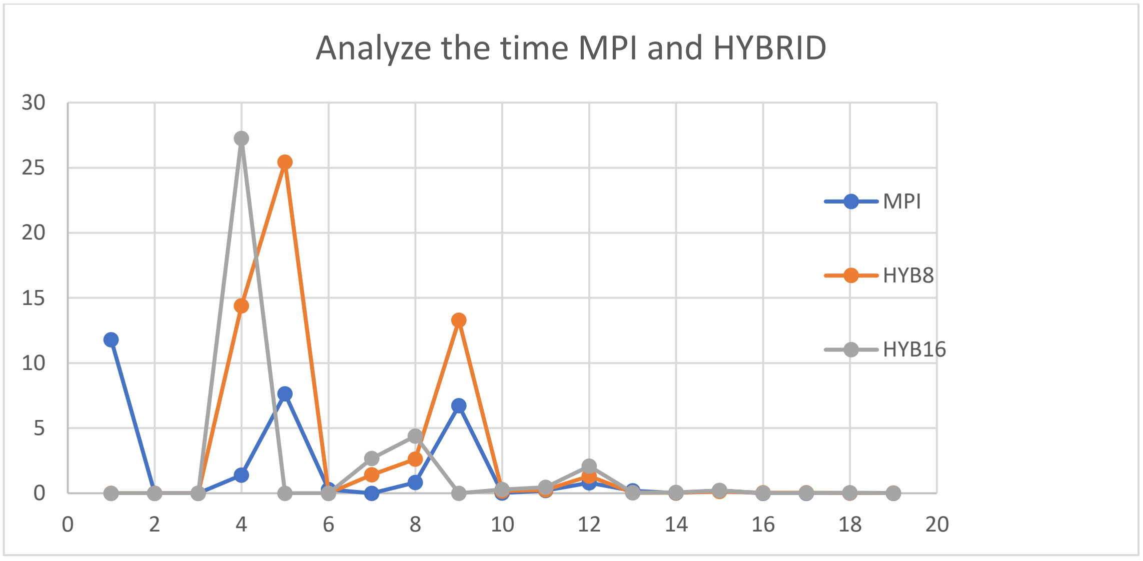

Figure 5 - Graph time MPI and Hybrid

In this graph we realize that in compare with the MPI and HYB8Threads we seem has the biggest difference when we increase amount a lot of data and iteration but when the data is less has not a big the difference.

Reference

https://hpcc.usc.edu/support/documentation/running-a-job-on-the-hpcc-cluster-using-pbs/

https://hpc-forge.cineca.it/files/ScuolaCalcoloParallelo_WebDAV/public/anno-2013/advanced-school/IntroMPI+OpenMP.pdf